University of British Columbia
Electrical and Computer Engineering
Electrical and Biomedical Engineering Design Studio
ELEC291/ELEC292

The MSP Microcontroller System

Introduction

This document introduces a minimal microcontroller system using TI's MSP430G2553 microcontroller. The MSP430G2553 is a 16-bit microcontroller, 16MHz, 16KB (8K x 16) FLASH, in a DIP20 package.

Recommended documentation

MSP430 User's Guide: http://www.ti.com/lit/ug/slau144j/slau144j.pdf

MSP430G2x53 Family Data Sheet: http://www.ti.com/lit/ds/slas735j/slas735j.pdf

Assembling the Microcontroller System

Figure 1 shows the circuit schematic of the MSP430G2553 microcontroller system used in ELEC291/ELEC292. It can be assembled using a bread board. Table 1 below lists the components needed to assemble the circuit.

Quantity	Digi-Key Part #	Description
3	BC1148CT-ND	0.1uF ceramic capacitors
2	BC1157CT-ND	1uF ceramic capacitor
2	270QBK-ND	270Ω resistor
1	1.0KQBK-ND	1kΩ resistor
1	100KQBK-ND	100kΩ resistor
1	MCP1700-3302E/TO-ND	IC REG LINEAR 3.3V 250MA TO92-3
1	67-1108-ND	LED 5MM GREEN
1	300-8842-ND	CRYSTAL 32.7680KHZ 7PF T/H
1	N/A	BO230XS USB adapter
1	296-28429-5-ND	MSP430G2553
1	P8070SCT-ND	Push button switch

Table 1. Parts required to assemble the MSP430G2553 microcontroller system.

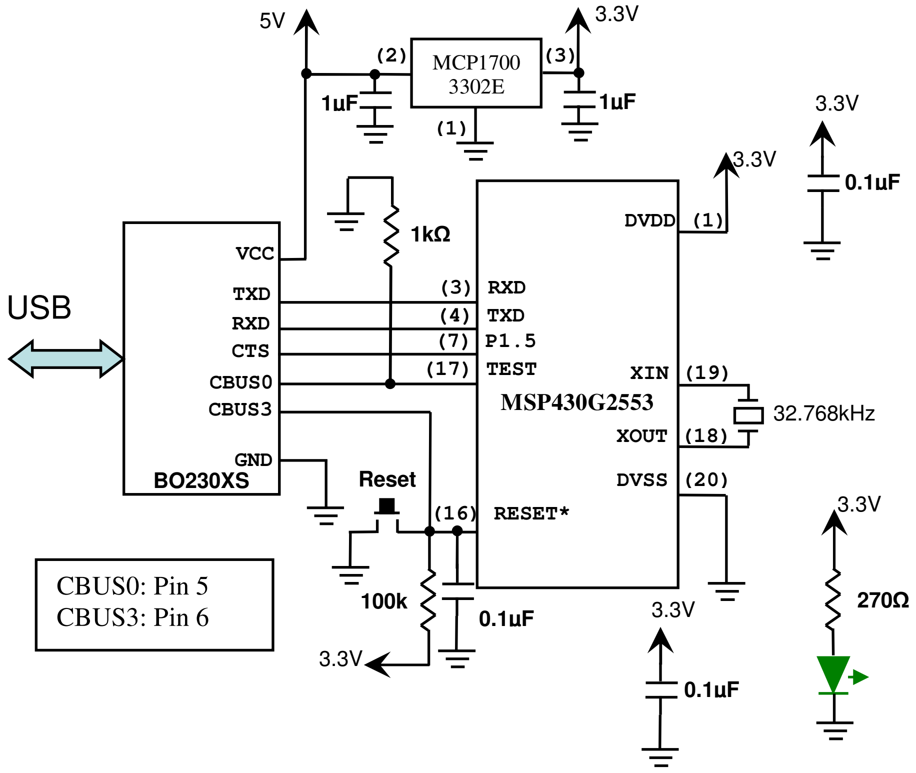


Figure 1. Circuit schematic of the microcontroller system.

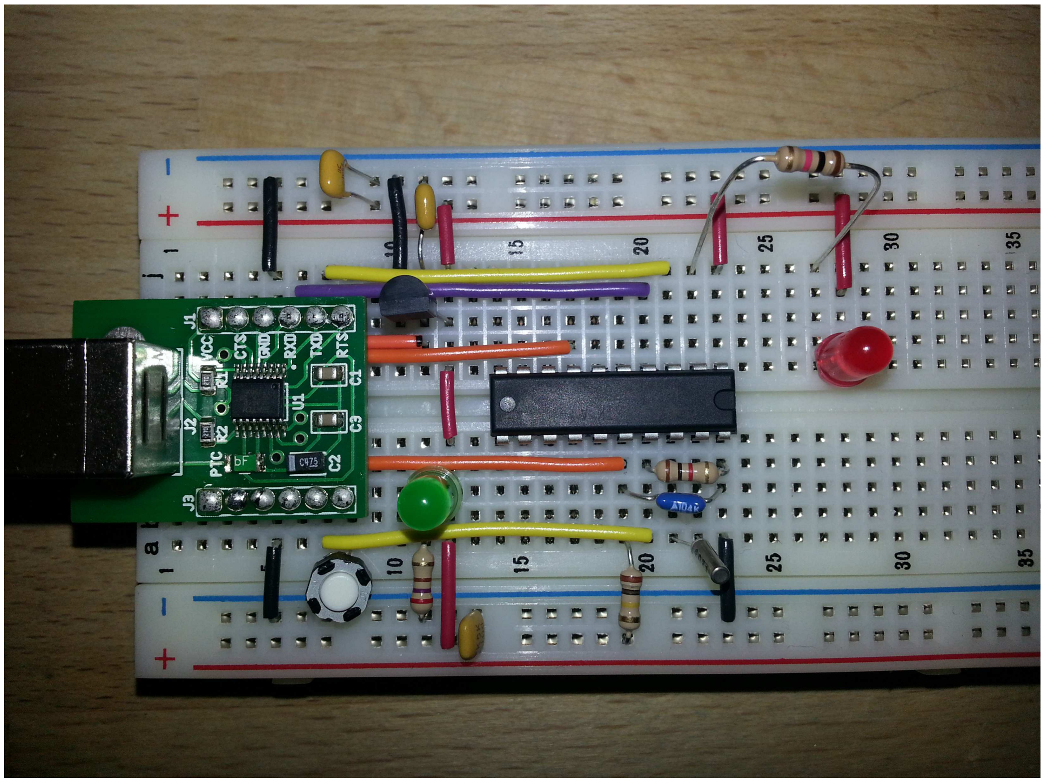


Figure 2. Bread boarded MSP430G2553 microcontroller system. The red LED is connected in series with a 1kΩ resistor to pin 2 (P1.0) as needed by the 'Blinky' example below.

Setting up the Development Environment

To establish a workflow for the ATMEGA328P we need to install the following three packages:

1. **CrossIDE V2.24 (or newer) & GNU Make V4.2 (or newer)**

   Download CrossIDE from: http://ece.ubc.ca/~jesusc/crosside_setup.exe and install it. Included in the installation folder of CrossIDE is GNU Make V4.2 (make.exe, make.pdf). GNU Make should be available in one of the folders of the PATH environment variable in order for the workflow described bellow to operate properly. For example, suppose that CrossIDE was installed in the folder "C:\crosside"; then the folder "C:\crosside" should be added at the end of the environment variable "PATH" as described here[1].

   Some of the Makefiles used in the examples below may use a "wait" program developed by the author. This program (and its source code) can be downloaded from the course web page and must be copied into the CrossIDE folder or any other folder available in the environment variable "PATH".

2. **GCC MSP430 Toolchain for Windows.**

   Download the GCC MSP430 Embedded Tool Chain from the link below. You'll need to download both the GCC compiler and the "Header and Support Files". The site will require you to register with TI.

   http://www.ti.com/tool/msp430-gcc-opensource

   Alternatively you can download from these two links:

   http://courses.ece.ubc.ca/281/2017/msp430-gcc-6.2.1.16_win32.zip
   http://courses.ece.ubc.ca/281/2017/msp430-gcc-support-files-1.198.zip

   Once the download is complete, decompress the archives somewhere in your computer. You'll need to adjust the provided example makefiles to reflect the correct path of the GCC compiler and support files from the links above.

3. **MSP430G2553 Flash Loader: MSP430_prog.**

   Available in the web page for the course is the program "MSP430_prog.exe" developed by the author. This program is included together with some examples (listed below) on programming the MSP430G2553 microcontroller. The provided example makefiles use this program to load the compiled program into the MSP430G2553 flash memory. You'll need to edit the makefiles to reflect the proper path of the programmer.

[1] http://www.computerhope.com/issues/ch000549.htm

Workflow.

The workflow for the MSP430G2553 microcontroller includes the following steps.

1. Creation and Maintenance of Makefiles.

CrossIDE version 2.24 or newer supports project management using **simple** Makefiles by means of GNU Make version 4.2 or newer. A CrossIDE project Makefile allows for easy compilation and linking of multiple source files, execution of external commands, source code management, and access to microcontroller flash programming. The typical Makefile is a text file, editable with the CrossIDE editor or any other editor, and looks like this:

```
# Since we are compiling in windows, select 'cmd' as the default shell.  This
# is important because make will search the path for a linux/unix like shell
# and if it finds it it will use it instead.  This is the case when cygwin is
# installed.  That results in commands like 'del' and echo that don't work.
SHELL=cmd

OBJS=blinky.o

# The paths of the GCC and support files downloaded from the internet
GCC_DIR =  C:/Programs/msp430_gcc/bin
SUPPORT_DIR = C:/Programs/msp430_gcc/include

DEVICE  = msp430g2553
CC      = $(GCC_DIR)/msp430-elf-gcc
LD      = $(GCC_DIR)/msp430-elf-ld
OBJCOPY = $(GCC_DIR)/msp430-elf-objcopy

CCFLAGS = -I $(SUPPORT_DIR) -mmcu=$(DEVICE) -O2 -g
LDFLAGS = -L $(SUPPORT_DIR)

# The default 'target' (output) is blinky.elf and 'depends' on
# the object files listed in the 'OBJS' assignment above.
# These object files are linked together to create blinky.elf.
# The linked file is converted to hex using program objcopy.
blinky.elf: $(OBJS)
	$(CC) $(OBJS) $(CCFLAGS) $(LDFLAGS) -o blinky.elf
	$(OBJCOPY) -O ihex blinky.elf blinky.hex
	@echo done!

# The object file blinky.o depends on blinky.c. blinky.c is compiled
# to create blinky.o.
blinky.o: blinky.c
	$(CC) -c $(CCFLAGS) blinky.c -o blinky.o

clean:
	del $(OBJS) *.elf *.hex

# You may add the location of MSP430_prog to the path or provide a full
# path for the program as shown below.
program: blinky.hex
	..\programmer\MSP430_prog -p -r blinky.hex

verify: blinky.hex
	..\programmer\MSP430_prog -v -r blinky.hex
```

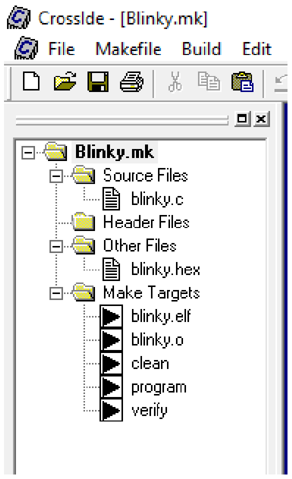
The preferred extension used by CrossIDE Makefiles is ".mk". For example, the file above is named "blinky.mk".

Makefiles are an industry standard. Information about using and maintaining Makefiles is widely available on the internet. For example, these links show how to create and use simple Makefiles.

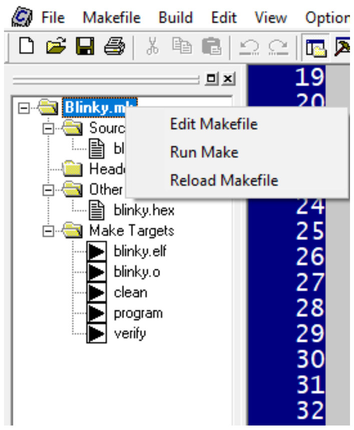
https://www.gnu.org/software/make/manual/make.html
http://www.cs.colby.edu/maxwell/courses/tutorials/maketutor/
https://www.cs.swarthmore.edu/~newhall/unixhelp/howto_makefiles.html
https://en.wikipedia.org/wiki/Makefile

2. Using Makefiles with CrossIDE: Compiling, Linking, and Loading.

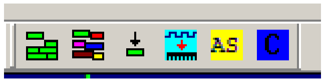
To open a Makefile in CrossIDE, click "Makefile"→"Open" and select the Makefile to open. For example "Blinky.mk". The project panel is displayed showing all the targets and source files:



Double clicking a source file will open it in the source code editor of CrossIDE. Double clicking a target 'makes' that target. Right clicking the Makefile name shows a pop-up menu that allows for editing, running, or reloading of the Makefile:



Additionally, the Makefile can be run by means of the Build menu or by using the Build Bar:



Clicking the 'wall' with green 'bricks' makes only the files that changed since the last build. Clicking the 'wall' with colored 'bricks' makes all the files. Clicking the 'brick' with an arrow, makes only the selected target. You can also use F7 to make only the files that changed since the last build and Ctrl+F7 to make only the selected target.

Compiling & Linking

After clicking the build button this output is displayed in the report panel of CrossIDE:

```
----------------- CrossIde - Running Make -----------------
C:/Programs/msp430_gcc/bin/msp430-elf-gcc -c -I C:/Programs/msp430_gcc/include -
mmcu=msp430g2553 -O2 -g blinky.c -o blinky.o
C:/Programs/msp430_gcc/bin/msp430-elf-gcc blinky.o -I C:/Programs/msp430_gcc/include -
mmcu=msp430g2553 -O2 -g -L C:/Programs/msp430_gcc/include -o blinky.elf
C:/Programs/msp430_gcc/bin/msp430-elf-objcopy -O ihex blinky.elf blinky.hex
done!
```

Loading the Hex File into the Microcontroller's Flash Memory

To load the flash memory to the microcontroller, double click the 'program' target. This output is then displayed in the report panel of CrossIDE:

```
----------------- CrossIde - Running Make -----------------
..\programmer\MSP430_prog -p -r blinky.hex
MSP430 programmer using BO230X board. (C) Jesus Calvino-Fraga (2017)
blinky.hex: Loaded 944 bytes.
Connected to COM12
Erasing main memory.  Please wait... done.
MSP430x2553 detected.  BSL Version: 2.03
Loading flash memory:
.................................... Done
Actions completed in 3.2 seconds.
```

A file named "COMPORT.inc" is created after running the flash loader program. The file contains the name of the port used to load the program, for example, in the example above COM4 is stored in the file. "COMPORT.inc" can be used in the Makefile to create a target that starts a PuTTy serial terminal session using the correct serial port:

```
PORTN=$(shell type COMPORT.inc)
.
.
putty:
	@Taskkill /IM putty.exe /F 2>NUL | wait 500
	c:\putty\putty.exe -serial $(PORTN) -sercfg 115200,8,n,1,N -v
```

For more details about using "COMPORT.inc" check the project examples below.

Project Examples

The following Project examples are available in the web page of the course.

blinky: 'blinks' an LED connected to pin 2 (P1.0). This is the same project used in the examples above.

blinky_isr: 'blinks' an LED connected to pin 2 (P1.0) every second using interrupts. This example uses Timer0 CC0 and CC1 to generate two separated interrupts. The interrupt associated with Timer0 CC0 toggles P1.0 every 0.5s while the interrupt associated with Timer0 CC1 toggles P2.1 every 0.25s.

tone: Uses Timer0 CC0 interrupt to generate a requested frequency. The desired frequency is input using the serial port with PuTTy.

ADC: Configures the ADC and serial port. It reads the voltage at pin 5 (input A3). Transmits the converted value using a custom made printf() function serially to PuTTy. Notice that this program uses almost all the memory available in the MSP430G2553 microcontroller, so it is not advisable to use the printf() provided to print floating point numbers. By the way, the printf() that comes with GCC is even worst; it won't fit in the memory of the MSP430G2553.

ADC_eff1: Configures the ADC and serial port. It reads the voltage at pin 5 (input A3). Transmit the converted value using a custom made _ftoa() function serially to PuTTy. This program is more efficient than the previous one but still consumes a fare amount of memory.

ADC_eff2: Configures the ADC and serial port. It reads the voltage at pin 5 (input A3). Transmit the converted value using only integer arithmetic and send the result serially to PuTTy. This program is more efficient than the two previous versions and it is advisable to follow this approach when writing your programs.

Uart: Configures and uses the uart together with PuTTy to transmit and receive characters. This example also shows got to setup an Interrupt Service Routine (ISR) for uart transmission, although I don't clearly understand what is going on. For interrupts it seems to be better to use the approach shown in the 'blinky_isr' example above.

SoftwareUart: Show how to implement a software uart using time delays.

ShowCalibration: Shows the factory stored calibration data in the MSP430G2553 via PuTTy.